OPTIMAL BOOLEAN SET OPERATION GENERATION AMONG POLYGON-REPRESENTED REGIONS

CROSS REFERENCE TO RELATED APPLICATIONS

[0001]   The present application derives priority from Provisional Patent Application No. 60/153,569 filed on 13 September 1999.

BACKGROUND

1. Field of the invention

[0002]   The present invention relates to Boolean set operations on data sets and, more particularly, to an improved method and software process for Boolean set intersection and set union operations among polygon-represented data sets using a digital computer.

2. Description of related art

[0003]   Boolean set operation generation is a foundational computational capability in a wide range of diverse problem domains, including image processing, spatial data analysis, constraint-based reasoning, earth resource evaluation, crop management, market analysis studies, micro-fabrication, mining, weather forecasting, military planning, and utility management. For example, rain forest shrinkage over time can be studied by performing Boolean set intersections between processed earth resource imagery and historical vector-represented geo-spatial map products depicting vegetation. The determination of regions "within 100 miles of Oklahoma City possessing a slope less than 3 degrees where wheat is grown" can be determined by performing Boolean set intersections between a circular region with radius 100 miles about Oklahoma City, a map product depicting slopes, and a land use product depicting agricultural crops. Ground-based target locations for military applications can often be significantly refined by intersecting sensor generated error ellipses with domain features that favor the presence of such vehicles (roads, high mobility regions, and regions that afford cover and concealment), while de-emphasizing regions that do not favor such vehicles (swamps and waterways). Each of these examples relies on Boolean set operation computation among potentially large 2-D

spatially-organized data sets. All Geographic Information Systems (GIS) and modern database management systems (DBMS) provide direct support to two-dimensional Boolean set operation generation.

[0004]   For polygon-represented data sets, the key Boolean set operations of intersection and union have been typically generated using algorithms from a field of mathematics known as computational geometry. Due to the highly combinatorial nature of these methods, computational requirements tend to increase rapidly as a function of the "size' of the data sets (i.e., the number of vertices in the polygon representations). Further dramatic increases in computational requirements occur when the polygons are multiply-connected (e.g., polygons with included holes or regions that consist of a collection of non-intersecting smaller regions). As a direct result of the inherent combinatorial computational requirements, large-scale dynamic applications have tended to be impractical. Even highly parallel implementations, as well as significant improvements in hardware performance cannot effectively overcome the staggering computational requirements of some prospective applications.

[0005]   Traditionally, two-dimensional objects could be represented by one of two generally accepted spatial data structures: either the region quadtree or tuple-based polygon representations. The region quadtree spatial data structure provides a regular, recursive decomposition of 2-D space. Quadtrees are used in the construction of some multidimensional databases (e.g., cartography, computer graphics, and image processing). A quadtree is a means of encoding an image as a tree structure. Each node of the tree has up to four children. The root node represents the entire image; its children represent the four quadrants of the entire image; their children represent the sixteen subquadrants, and so on. Because the region quadtree is an areal-based representation, it supports highly efficient top-down spatial search and data manipulation, as well as areal-based set operation generation.

[0006]   Despite these benefits, the region quadtree representation exhibits significant shortcomings. Because the "size" of the quadtree representation increases by a factor of four for each level of decomposition, excessively large storage requirements (and concomitant computational requirements) can occur for applications that demand high fidelity data representation. Tuple-based representations, on the other hand, provide memory-efficient, high

2

fidelity representations of point, line, and region features. However, because such representations do not directly support spatial search and spatial data manipulation, data analysis tends to lead to highly combinatorial computational requirements.

[0007]   Thus, both tuple-based and region quadtree-based spatial data representations possess strengths, as well as obvious weaknesses. Spatial search, reasoning, and set generation that tends to be straightforward and efficient for quadtree-based representations, but lead to computationally-intensive algorithms for tuple-based representations. Data storage requirements to support high fidelity data sets are readily met with tuple-base representations, while direct use of region quadtree representations leads to excessively large storage and associated data manipulation requirements.

[0008]   There are few prior efforts at more efficient set operation generation procedures. One example, U.S. Patent No. 5649084, discloses a method for performing Boolean operations on geometric objects in a computer-aided design system. Edges of a first object are intersected with surfaces of a second object to produce intersection points, and surfaces containing the faces of the two objects, respectively, are intersected with each other to produce intersection tracks. If there is an inconsistency between the intersection points and corresponding intersection tracks, a perturbation step is applied to correct the spatial positions of inconsistent intersection points. This maintains geometric consistency and avoids overly complex set operations. However, it also changes the original objects and sacrifices accuracy.

[0009]   Rather than seeking incremental enhancements over traditional approaches, in the early 1980s the present inventor pioneered an entirely new data representation structure referred to as the region quadtree-indexed vector representation. See, Antony, R. and Emmerman, P., Spatial Reasoning and Knowledge Representation, Geographic Information Systems in the Government Workshop Proceedings, A. Deepak Publishing (1986). This structure helps to dramatically reduce the computational requirements for set operation generation among both simple and multiply-connected polygon-represented data sets. The quadtree-indexed vector representation is, in effect, a hybrid data structure that combines the benefits of a hierarchical, multiple resolution grid-based spatial representation (the region quadtree data structure) with the accuracy and storage efficiency of tuple-based (polygon boundary) representations. In the quadtree-

indexed vector representation, the region quadtree serves not only as a spatial indexing mechanism into memory-efficient, high fidelity representations of points, lines, and region boundaries, but as a multiple resolution spatial representation in its own right.

[0010]   Fig. 1 illustrates the relationship between an indexing cell(s) and the underlying tuple-based data. Note that for point and line (commonly referred to as polyline) features, every quadtree cell in the representation "indexes" tuple-based data. For regions, however, there exist two classes of quadtree indexing cells: those that index interior cells and those that index boundary cells. Interior cells are quadtree cells that are fully included within a region; boundary cells are partly within and partly outside a region. Thus, for simply-connected regions, only quadtree boundary cells associated with the region's boundary directly index tuple-based data.

[0011]   For polygons that possess one or more holes as shown in Fig. 2, tuple lists define not only the outer boundary of the polygon, but also the boundaries of all included holes. The tuple list of a hole serves both as the description of the edge of the hole, as well as a description of an inner edge of the region. The "direction" associated with the tuple list of a first-order hole is opposite that of the outer boundary list convention. For embedded holes, each subsequent tuple list is order-reversed to maintain logical consistency with respect to the "inside" and "outside" of the region.

[0012]   Fig. 3 illustrates tuple-represented piece-wise continuous line feature representing a region boundary along with the associated quadtree-indexing cells. This figure shows the association between line segment vertex tuples and the quadtree-indexing cell (actual data points are shown as filled circles; pseudo points are shown as open circles). The outer square in the figure represents an indexing cell at some arbitrary level of decomposition. Data points 2, 3, 4, 5, and 10 are seen to lie within the cell. The quadtree-indexed vector spatial data structure requires the addition of pseudo points at all cell entrance and exit boundaries. Referring back to Fig. 3, four pseudo points are associated with this particular cell: points 1, 6, 9, and 11. If represented at the next higher resolution quadtree grid size (indicated by the four smaller squares in Fig. 3), additional pseudo points are required at the boundaries of the smaller indexing cells. For example, the upper right indexing cell consists of the entrance tuple pseudo point 3', the data tuples 4 and 5, and the exit tuple pseudo point 6. Because one or more data tuples could fall on a

cell boundary (a data tuple actually lies at the cell boundary or a line segment follows the cell boundary), the implementation must distinguish pseudo points from the original tuple list. Adjacent cells maintain duplicate shared cell boundary tuples. The lower right and the upper right cells, for example, share tuple 3'.

[0013]   In addition to distinguishing between data point tuples and indexing cell boundary pseudo points, the implementation must distinguish between interior (non-boundary) indexing cells and boundary indexing cells.

[0014]   Figs. 4(a)-(c) shows the interaction between two regions, and Figs. 4(d)-(f) shows the interaction between the region's respective quadtree grids. Unless two regions share at least one common quadtree indexing cell, no direct interaction between the two data sets occur. When no common cells occur, set intersection is null and set union is the set formed by all the cells in the two disjoint regions. However, when common cells do occur, the set intersection is as seen in Fig. 4(f).

[0015]   Fig. 5 illustrates a categorical approach for analyzing set operations that was developed by the inventor herein and is described in Antony, R., Principles of Data Fusion Automation (Artech House, 1995). When set intersection exists, set operation generation is treated as a three-stage (sub-problem) process involving the following canonical form classes.

[0016]   Class 1: interactions between two interior cells;

[0017]   Class 2: interactions between a boundary and an interior cell; and

[0018]   Class 3: interactions between two boundary cells.

[0019]   The first two stages entail only relatively trivial computations, and the appropriate methods for generating the products from both Class 1 and Class 2 are described in the Antony treatise, supra at 95. The third stage, however, is considerably more involved and effectively controls the computational complexity of the set operation generation process. Because both polylines and points can be treated as degenerate regions (regions with no "interior cells"), set union and intersection operations for these lower order features can be treated as a special case of the region generation methodology.

[0020]   Consequently, it would be greatly advantageous to provide an optimal method for generating the product of the third, and key stage of the procedure: Class 3 interactions between two boundary cells.

SUMMARY OF THE INVENTION

[0021]   It is, therefore, an object of the present invention to provide an improved method for generating Boolean set intersection and set union among polygon-represented data sets using a digital computer.

[0022]   It is another object to provide a method as described above which is computationally superior to current computational geometry methods, giving orders of magnitude improvement in efficiency over current computational methods.

[0023]   It is another object to provide a method of Boolean set operations that fully capitalizes on the efficiencies gained by quadtree-indexed vector representation of "tuple-based lists" that define region(s) associated with the union and intersection of the original data sets.

[0024]   According to the present invention, the above-described and other objects are accomplished by providing a method and software implementation for computing Boolean set operations on two regions defined by a quadtree-indexed vector representation of data point tuples. The method steps initially include establishing indexing cells about the two regions.

[0025]   Next, the method distinguishes the three canonical form classes: Class 1 interactions-indexing cells containing only interior portions of both regions (interior, interior); Class 2 interactions- indexing cells containing only interior portions of one region and boundary portions of the other region (interior, boundary); and Class 3 interactions - indexing cells continaing a portion of the boundaries of both of the regions (boundary, boundary). As described above, the first two stages entail only relatively trivial computations, and the appropriate methods for generating the products from both Class 1 and Class 2 are described in the Antony treatise, supra at 95. The present method focuses on the third Class 3 which is considerably more involved and effectively controls the computational complexity of the set operation generation process.

[0026] As a next step, pseudo points are defined for each (boundary, boundary) indexing cell. Four separate pseudo points are defined for each (boundary, boundary) indexing cell, one where each of the two regions enter and exit the indexing cell.

[0027] Next, each (boundary, boundary) indexing cell is categorized based on a relationship of its defined pseudo points, and one of the two regions is selected to be a starting region for that indexing cell based on the categorization.

[0028] Given the category and starting region, the set operation is performed by including a list of tuples in each (boundary, boundary) indexing cell that are encountered while following ("tracing") the boundary of the starting region until an intersection of the boundaries of said two regions occurs, then accumulating tuples associated with the other of region, and so on until the indexing cell is traversed.

[0029] Given the tuples included from the set operations on all (boundary, boundary) indexing cells, it is a simple task to combine the result with the set operations on (interior, interior) indexing cells and (interior, boundary) indexing cells in a known manner to produce a complete union or intersection.

[0030] The foregoing method fully capitalizes on the efficiencies gained by quadtree-indexed vector representation, and as shown below is capable of giving orders of magnitude improvement in efficiency over current computational methods.

DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING(S)

[0031] Other objects, features, and advantages of the present invention will become more apparent from the following detailed description of the preferred embodiment and certain modifications thereof when taken together with the accompanying drawings in which:

[0032] Fig. 1 is a graphical illustration of interior and boundary quadtree cells associated with a polygon-represented region;

[0033] Fig. 2 is a graphical example of a multiply-connected polygon having a hole;

[0034] Fig. 3 illustrates a two-level decomposition of a quadtree-indexed vector spatial representation;

[0035] Fig. 4(a)-(f) illustrates an example of the quadtree-based intersection of two regions;

[0036] Fig. 5 is an exploded view of the set operation interaction shown in Fig. 4;

[0037] Fig. 6 is a flow chart implementation reflecting the software steps for implementing the software method of the present invention;

[0038] Fig. 7 is a graphical illustration of the intersection of a boundary cell and an interior cell (upper row a-c) and the intersection of two boundary cells (lower row d-f). The interior of all regions are shown as shaded;

[0039] Fig. 8 is a graphical illustration of the various kinds of bounding box relationships between line segments for two features;

[0040] Fig. 9 is a graphical example of two lineal features within a quadtree indexing cell showing each tuple-pair bounding box;

[0041] Fig. 10 is a graphical illustration of the four possible interactions between two line segments whose bounding boxes intersect;

[0042] Fig. 11 is a graphical illustration of two region boundary features in two adjacent boundary cells;

[0043] Fig. 12 is a graphical illustration of two region boundary features that both reenter a cell;

[0044] Fig. 13 is a graphical illustration showing a feature boundary reentry;

[0045] Fig. 14 is a catalog of twenty-four possible boundary cell interaction cases;

[0046] Fig. 15 is a table that summarizes the prescribed manner for following (or "tracing") the features from a designated entry or exit point and onward to the next polyline intersection point;

[0047] Fig. 16 is a graphical illustration of a grid of nine quadtree-indexing cells associated with two polygon-represented regions;

[0048] Figs. 17 and 18 provide expanded views of the center cell of Fig. 16;

[0049] Figs. 19-22 provide graphical illustrations of representative examples that demonstrate the application of Fig. 15 for Categories III – VI;

[0050] Fig. 23 is a graphical illustration of a representative example Category I interaction;

[0051] Fig. 24 is a graphical illustration of a representative example Category II interaction;

[0052] Figs. 25-28 provide graphical examples that illustrate the analysis of regions with included holes;

[0053] Fig. 29 is a table that summarizes general characteristics of each canonical form class; and

[0054] Fig. 30 is a graphical illustration of two regions exhibiting (a) limited boundary cell overlap and (b) two regions exhibiting extensive edge overlap;

DETAILED DESCRIPTION OF THE INVENTION

[0055] An entirely new and improved method and software implementation is disclosed for generating Boolean set intersection and set union among two regions defined by a quadtree-indexed vector representation of data point tuples using a digital computer. Rather than relying on traditional cell-by-cell ANDing operations for identifying indexing cells that interact, the present method treats index cell interaction as a spatial search problem.

[0056] Two of the steps are known (Antony treatise, supra at 95), and these include the steps of: 1) establishing indexing cells about the two subject regions; and 2) distinguishing three canonical form classes of indexing cells: Class 1 interactions containing overlapping interior cells of both regions; Class 2 interactions containing interior cells of one region and a portion of a boundary of the other regions; and Class 3 interactions which contain a portion of the boundaries of both of the regions (boundary, boundary). The remaining novel steps, when

9

performed in combination, speed set operation processing considerably. These steps are generally as follows: 3) defining pseudo points for each Class 3 (boundary, boundary) indexing cell; 4) categorizing each (boundary, boundary) indexing cell based on a relationship of its defined pseudo points, and identifying one of the two regions to be a starting region for that indexing cell based on the categorization; and 5) given the category and starting region, performing the set operation by including all tuples in each (boundary, boundary) indexing cell that are encountered while following ("tracing") the boundary of the starting region until an intersection of the boundaries of the two regions occur, then accumulating tuples associated with the other region, and switching regions at each intersection until the indexing cell is traversed.

[0057]   Fig. 6 is a flow chart implementation reflecting the software steps for implementing the above-described method.

[0058]   At step 10, the method initializes the bounding box and then establishes a grid of indexing cells about two subject regions. Initialization of the bounding box is simply accomplished by establishing the problem space about two subject regions, and then decomposing the problem space into the grid array of indexing cells. At the first level of decomposition, a selected problem space is sub-divided into four equal-sized quadrants. Each of these quadrants is then recursively decomposed into four equal size quadrants. Preferably, the number of levels of decomposition can be user-selected. At each level of decomposition, the quadtree provides a mutually exclusive and exhaustive representation of a selected region of the problem space. The net result is a variable (user-sized) grid of indexing cells about the two subject regions.

[0059]   At step 20, pseudo-points are defined for the borders of the two regions. For each indexing cell of the grid that contains part of the boundary of at least one region (a ~~Boundary~~boundary indexing cell), pseudo-points are assigned at the position of entry and exit of each region's border from that indexing cell.

[0060]   At step 30, the desired type of set operation is retrieved (by user-input or preprogramming), and this may either be Quad intersection or union.

[0061]   At step 40, the vector operation is performed. The software implementation of this step preferably involves two precautionary pre-tests to avoid undue processing. A first pre-test is run to ensure that the operation is not being performed on the same region (as opposed to two different regions). If performed on the same region, the solution for both indexing cell (Quad) intersection or union is simply the list of tuples associated with the singular region. The second pre-test is performed to ensure that the solution is not a null solution (in other words, that the two regions do interact by some degree).

[0062]   Assuming some interaction, the operation continues to the novel processing steps in accordance with the present invention. Interactions between indexing cells are classified as follows: interactions between two ~~Interior~~interior cells (interior, interior), interactions between a ~~Boundary~~boundary and an ~~Interior~~interior cell (interior, boundary), and interactions between two ~~Boundary~~boundary cells (boundary, boundary) indexing cells. These three classes are treated separately as follows:

[0063]   <u>Class 1: (interior, interior) cells</u>

[0064]   All matching interior cells within the two regions are added to the intersection product. For set union, all (non-duplicate) interior cells of the two regions are included in the product. Duplicate cells can be readily identified as those interior cells that are within the set intersection product (to be within the intersection product, both regions must necessarily contain identical cells). If the cells in the two data sets possess different indexing cell sizes (i.e., different levels of quadtree decomposition), the product for set intersection is the smaller of the two cells, while the product for set union is the larger of the two cells. Consequently, both Class-1 intersection and union generation operations inherently handle disparate size quadtree cells.

[0065]   <u>Class 2: (interior, boundary) cells</u>

[0066]   As illustrated in the first row of Fig. 7 (a-c), the set intersection of a boundary cell (a) and an interior cell (b) yields the boundary cell (c), unless the boundary cell is physically larger (coarser decomposition level) than the interior cell. In this case, the intersection product is the boundary representation that occurs within the smaller interior quadtree cell. For set union, the product is the interior cell (b), independent of cell size.

[0067]   Class 3: (boundary, boundary) cells

[0068]   Whereas Class 1 and Class 2 set operation products are straightforward to compute, Class 3 products (second row in Fig. 7) represent a considerably more challenging problem. The method of the instant invention includes an optimal procedure for generating Class 3 Boolean set operation products. The optimal procedure stems from the realization that for set operation generation, there exist six categorical Class 3 product forms. These forms are topologically complete in the sense that they characterize all possible region boundary interactions between two data sets. Moreover, given the canonical form category of a particular (boundary, boundary) cell, the set operation procedure for that cell can be determined based solely on the ordering of the entrance and exit pseudo-points along the boundary of a quadtree-indexing cell. This characterization leads to a well-behaved procedure capable of efficiently constructing both the intersection and union product for a given indexing cell once (1) the canonical form category is established and (2) all polyline feature intersection points within the cell are computed. As proven below, the procedure is potentially order of magnitudes faster that traditional computational geometry approaches. Furthermore, since each (boundary, boundary) cell operation is fully independent, Class 3 operations can be performed in parallel, further enhancing set operation generation performance.

[0069]   Specifically, the Class 3 operation begins by employing simple algebraic techniques to compute all intersection points between the linear boundaries of the two regions within an indexing cell. Once computed, these intersection points are added at their appropriate location within each feature's ordered tuple lists.

[0070]   Fig. 8 is a graphical illustration of the various kinds of bounding box relationships between line segments for two features.

[0071]   Fig. 9 is a graphical example of two lineal features within a quadtree indexing cell showing each tuple-pair bounding box. Arrowheads indicate line segment direction.

[0072]   With collective reference to Figs. 8 and 9, note that the intersection points must be tagged to distinguish them from both data tuples and pseudo points. This is necessary because bounding boxes associated with individual tuple pairs from two (or more) regions might intersect

while the line segments (represented by a tuple pair) might not intersect. In the example shown in Fig. 8, the bounding boxes for some ~~black/bold~~solid segments (indicated in Fig. 8 by the "prime" mark " ' " in the top portion of the figure) and ~~gray/unbold~~dashed segments (indicated in Fig. 8 by unprimed numbers in the top portion of the figure) overlap while others do not (For convenience, the terms "~~black~~solid" and "~~gray~~dashed" are used throughout this document to distinguish the two features that exist within a given indexing cell. However, the designation is totally arbitrary). Because line segments whose bounding boxes overlap may not actually intersect, bounding box intersection is merely a precursor to line segment intersection determination.

[0073]   A simple procedure, such as the approach offered below, can be used to compute all line segment intersection points within a boundary cell:

[0074]   1. Test all line-segment bounding boxes associated with tuple pairs for possible intersection.

[0075]   2. For all bounding box pairs that intersect, determine possible intersection points of the line segments by first computing the intersection points of the associated lines (Note: these lines extend to infinity):

$y = mx - mx_i + y_i$ (associated with ~~gray/unbold~~dashed segment), and

$y' = m'x' - m'x'_i + y'_i$ associated with ~~black/bold~~solid segment);

where $m = (y_{i+1} - y_i)/(x_{i+1} - x_i)$,

$m' = (y'_{i+1} - y'_i)/(x'_{i+1} - x'_i)$,

$y = mx - mx_i + y_i = mx + K$ where $K = -mx_i + y_i$, and

$y' = m'x = m'x'_i + y'_i = m'x + K'$ where $K' = -m'x'_i + y'_i$;

set $x = x'$ and $y = y'$ to locate point of intersection:

$\Rightarrow (m - m')x = K' - K$; and

$\Rightarrow x$ intersection point $= (K' - K)/(m - m')$.

[0076]   To prevent overflow, it is necessary to check to see if $m = m'$ before this computation is performed. If $m = m'$, do not attempt to compute the intersection point since the two lines are parallel and therefore can not intersect. Once the x value of the intersection point is computed, then compute the y intersection point:

$$y \text{ intersection point} = \cancel{(K'-K)/(m-m')} = mx - mx_i + y_1$$

[0077]   3. To determine whether the line segments actually intersect, test whether the computed intersection point lies within the line-segment bounding boxes for both features. If the intersection point lies within both bounding boxes, the two line segments intersect, otherwise, they do not intersect.

[0078]   4. All discovered intersection points are then added to both feature tuple lists between the appropriate end point tuples.

[0079]   Fig. 10 illustrates the four possible interactions between two line segments whose bounding boxes intersect. In type 1 interaction, no intersection occurs. In type 2, line segments touch at a point. In type 3, line segments from the two features are co-linear. In type 4, two features cross each other at a point.

[0080]   Fig. 11 illustrates two region boundary features in two adjacent boundary cells. In this case, the ~~black/bold~~solid feature begins in the left-hand cell, moves to the right-hand cell, and then reenters the left-hand cell. In such case, the intersection product follows the ~~gray/unbold~~dashed (or "inside" feature) boundary.

[0081]   Fig. 12 illustrates two region boundary features that both reenter a cell. In this case, each cell is treated independently, and all secondary (tertiary, etc.) entrance points are handled sequentially. In the case of the left hand cell in Fig. 12, the boundaries marked "~~Black~~Solid 1" and "~~Gray~~Dashed 1" could be treated first, the product of "~~Black~~Solid 2" and "~~Gray~~Dashed 2" treated second, and the two results combined. Alternatively, the first product could be combined with "~~Black~~Solid 2" and the resultant product combined with "~~Gray~~Dashed 2." The basic analytical process described herein, however, remains the same.

[0082]   Fig. 13 illustrates a feature boundary reentry and demonstrates that the intersection product consists of two distinct closed line-segment "cycles" as outlined by the ~~black/~~bold ~~curves~~ line segments. The cycles include both intersection points and pseudo-points. A set operation is performed by including a list of tuples encountered while following ("tracing") the cycles until complete. However, the tracing of line-segments depends on the orientation of the region boundaries, which is in turn reflected by the orderings of entry and exit pseudo-points.

[0083]   Fig. 14 catalogs the twenty-four possible boundary cell interaction cases, providing a systematic and exhaustive characterization of all possible entrance and exit pseudo-point tuple orderings. Because, as already discussed, the starting tuple for the orderings is entirely arbitrary, four cyclical orderings exist that describe the identical entrance and exit point relationships. For instance:

[0084]   ~~Gray (unbold)~~Dashed Entrance (GD_E), ~~Black (bold)~~Solid Entrance (BS_E), ~~Black~~Solid exit (BS_X), ~~Gray~~Dashed exit (GD_X) characterizes the identical cell boundary relationships (and consequently represents the same canonical form) as: ~~Black~~Solid Entrance (BS_E), ~~Black~~Solid exit (BS_X), ~~Gray~~Dashed exit (GD_X), ~~Gray~~Dashed Entrance (GD_E). Consequently, there exist only six canonical orderings.

[0085]   In the method of the present invention, the set operation is completed by collecting tuples from the two region lists (in a manner prescribed by the canonical ordering of entrance and exit point relationships) until the cycle closes on itself (the sequence of assigned tuples returns to its starting tuple).

[0086]   More specifically, Class 3 union and intersection operations generate two types of tuple cycles: boundary closing and internal. Boundary-closing cycles are formed by starting at either a ~~black~~solid (BS_E) and/or (depending on the boundary cell x boundary cell category) ~~gray~~dashed (GD_E) feature entry tuple (or pseudo point). Boundary cycles are so named because the cycle implicitly closes along the cell boundary (or edge). Internal cycles begin at a polyline intersection point and follow features in a prescribed manner, until the initial intersection point is reached (or the cycle is complete). In both cases, the features are followed from a designated entry or exit point and onward to the next polyline intersection point of the two regions. The tracing then switches to the other region, accumulating tuples associated with the other region,

and so on until the indexing cell is traversed. This process of switching features at each successive polyline intersection point continues until the cycle is complete. A boundary-closing cycle is complete once the edge of the quadtree cell is reached (in other words, when a pseudo point on the indexing cell boundary is hit). Internal cycles are complete when the accumulation process reaches the starting tuple for that cycle (without encountering a pseudo point on the indexing cell boundary).

[0087]   Fig. 15 is a table that summarizes the prescribed manner for following (or "tracing") the features from a designated entry or exit point and onward to the next polyline intersection point, etc., and onward according to the generation methodology of the present invention. Column 1 lists the six unique cyclical orderings of entrance and exit tuples for a given quadtree-indexed cell. Column 2 identifies the assigned interaction Category ("interaction class") number (note that the categories are logically grouped and are not in strictly numeric order). Columns 3 and 4 specify the starting points for intersection and union generation, respectively, assuming "inside is to the right." Columns 5 and 6 specify the starting points for intersection and union generation, respectively, assuming "inside is to the left."

[0088]   This recursive cycle generation process continues until all polyline intersection points within the cell have been used. As indicated by Fig. 15, for Categories III-VI, both intersection and union cycle generation begins with a boundary-closing cycle (cycle begins at a pseudo point). However, this is not the case of Categories I and II. We describe the cycle generation process for Categories III-VI first and then discuss the necessary modifications required to accommodate Categories I and II.

[0089]   <u>Category III-VI Intersection/Union Product Generation</u>

[0090]   The principal steps in constructing Category III - VI (boundary, boundary) intersection and union products can be summarized as follows:

[0091]   (1) Evaluate the canonical form Category by determining the counter-clockwise ordering of the cell boundary entrance and exit points for the two features present in the cell. If one or both features reenter the cell, each interaction can be treated separately, or alternatively, the cell can be further decomposed to attempt to eliminate or at least reduce the number of feature re-entries, and thereby the complexity of the analysis.

[0092]   (2) Construct the appropriate boundary-closing cycle(s) by determining from Fig. 15 the appropriate cell entry point (either ~~gray~~<u>dashed</u> entrance or ~~black~~<u>solid</u> entrance) to begin cycle construction. Category III-VI possess only a single boundary closing cycle.

[0093]   (3) Because boundary-closing cycles close on the edge of the quadtree-indexing cell, processing for such a cycle terminates once a pseudo point on the cell boundary is encountered during the cycle traversal operation. The boundary-closing segments of a boundary-closing cycle represent implicit boundaries and are not actually reported as part of the set operation product.

[0094]   (4) Once the boundary-closing cycle has been constructed (the tuples collected), the next "unused" polyline intersection point (if one exists) associated' with the "entrance" feature list becomes the starting point for an internal cycle. Just as before, succeeding tuples for that feature are added to the list (effectively "tracing" the line segments in the region boundary) until the next internal polyline intersection point is reached. At that point, the tuple accumulation process ("feature-following") shifts to the other feature. This cycle-forming process continues until this new cycle is complete (the initial tuple in the cycle list is reached). Additional cycles, if they exist, begin at the next unused polyline intersection point.

[0095]   (5) This "cycle generation" process is repeated until all the polyline intersection points within the cell have been utilized. The generated tuple list(s) becomes the final set operation product for that cell.

[0096]   The (boundary, boundary) cell set operation generation process just described is further illustrated in the example shown in Fig. 16.

[0097]   Fig. 16 depicts a grid of nine quadtree-indexing cells associated with two polygon-represented regions. Cell 1 does not index either feature and therefore is not involved in any subsequent analysis. Cells 2, 3, 5, 6, 8, and 9 index the boundary of the ~~gray/unbold~~dashed feature; Cells 4-9 index the boundary of the ~~black/bold~~solid feature. Cells 2, 3, 4, and 7 index only a single feature and therefore contribute to the union product, but not to the intersection product. Cells 5, 6, 8, and 9 contain both ~~gray/unbold~~dashed and ~~black/bold~~solid feature tuples and therefore represent Class 3 interactions between two boundary cells (Antony treatise, supra at 95).

[0098]   To simplify the discussion, we focus our attention on only the center cell (Cell 5) in Fig. 16.

[0099]   Figs. 17 and 18 provide expanded views of the center cell of Fig. 16. Specifically, Fig. 17 illustrates the intersection product generation process; Fig. 18 illustrates union generation. The polyline intersection points (inserted into the working memory tuple lists for both features) are shown as solid black circles; the entrance and exit pseudo points are indicated by open circles. The counter-clockwise ordering of the entrance and exit tuples ($G\underline{D}_X$, $B\underline{S}_E$, $G\underline{D}_E$, $B\underline{S}_X$) represents a cyclical ordering of Category V (Fig. 15, column 1, row 6). Using the convention that the inside of a region is "to the right," column 3 of Fig. 15 indicates that for a Category V interaction, intersection generation requires a single boundary-closing cycle (as seen in last row, third column). As indicated by this entry, the intersection product begins with the ~~gray~~dashed entrance pseudo point $G\underline{D}_E$.

[00100] The desired set operation product for a cell is a list of all tuples for that part of the region intersection that occurs within the cell shown in Fig. 17. Starting with the ~~gray~~dashed entrance point tuple, the ~~gray~~dashed feature is followed (i.e., all encountered ~~gray~~dashed tuples in the list are added to the intersection product) until the first polyline intersection point occurs (labeled Point 2 in the figure). At the intersection point, the tuple accumulation process continues by accumulating tuples associated with the ~~black~~solid feature (Point 3 is added to the product) until the next intersection point occurs (Point 4). Traversal is always in ascending order

of the tuple lists. Because Point 4 is a polyline intersection point, accumulation continues by collecting tuples associated with the ~~gray~~dashed feature (Point 5 is added to the list) until the third intersection point is reached (Point 6). Tuple accumulation resumes by following the tuple list of the ~~black~~solid feature until the ~~black~~solid feature reaches the cell boundary. At cell exit Point 7, the boundary-closing cycle processing terminates. (As discussed previously, a boundary-closing cycle implicitly closes on itself by following the boundary edge in the appropriate direction until the starting point of the cycle is reached. However, boundary cell edges are not reported as part of the set operation product). Thus, the intersection product for Cell 5 is reported as the list of Points 1-7. The bold gray arrows in Fig. 16 trace the boundary closing cycle. The gray-shaded hatched region denotes the "implied" region intersection product.

[00101] Fig. 18 illustrates the set union product generation for the center cell of Fig. 16. Referring to Fig. 15, column 4, row 6, the union product is seen to begin with the ~~black~~solid entrance feature. Just as in set intersection generation, the union product is produced by accumulating tuples from the two feature tuple lists (to which have been added all appropriate pseudo points and embedded polyline intersection points), alternating between ~~black~~solid and ~~gray~~dashed tuples following each polyline intersection point. The union product is reported as the ordered (region boundary) list containing the seven points shown in Fig. 18. As is apparent, the only tuples shared by both the intersection and union products are the polyline intersection points. In this case, the union product is indicated by the ~~black~~cross-hatched region.

[00102] Figs. 19-22 provide further representative examples that demonstrate the application of Fig. 15 for Categories III - VI. Intersection points are numbered in the direction of their use for intersection product generation. When there are complex interactions between two sets, it is often easier to picture NOT Union (regions outside the union product). Thus, in the following examples, regions not contained in (i.e., outside) the union product are ~~outlined in bold black lines~~filled horizontally. Intersection cycles are ~~outlined in gray/thin line segments~~filled vertically. In other words, for set union, the entirety of each cell is inside the union product except for the union cycles ~~outlined by bold black lines~~filled horizontally.

[00103] Category I and II Intersection/Union Product Generation

[00104] Assuming "inside is to the right," for a Category 1 (boundary, boundary) cell interaction, set union begins with two boundary-closing cycles. For set intersection, on the other hand, there exist no boundary-closing cycles. The union boundary-closing cycles begin at both the ~~gray amid black~~dashed and solid feature entry points. For Category II, set intersection begins with two boundary-closing cycles. Set union contains no boundary-closing cycles. The intersection boundary-closing cycles begin with both the ~~gray and black~~dashed and solid entry points. Just as in earlier examples, the intersection cycles ~~shown in bold gray~~filled vertically depict regions of the cell that are included in the product, whereas ~~union cycles indicated in bold black~~horizontal fill depicts regions of the cell that are not included in the union product.

[00105] Examples of the two cell interaction Categories are shown in Figs. 23 and 24.

[00106] Fig. 23 offers an example of Category I interaction. As indicated in Fig. 1 5, for set union, boundary-closing cycles occur at both the ~~black~~solid and the ~~gray~~dashed feature cell entrance points; the union cycles are shown by the ~~bold black lines beginning at both G~~$_E$~~ and B~$_E$~~ horizontal fill. Because inside is to the right,~~ these black triangular-shaped regions represent area excluded from the union, depicting areas within the cell that are not included in either region. The internal cycles shown ~~outlined in bold black lines~~with horizontal fill represent additional cycles ("holes" in the union). Note that alternating with the ~~bold black~~horizontal fill cycles, are other cycles are outlined in ~~bold gray lines~~vertical fill. As in the earlier discussion, these cycles are associated with set intersection. As indicated in Fig. 15, set intersection cycles are generated by starting with either the ~~black~~solid or ~~gray~~dashed feature at the first polyline intersection point (rather than the cell boundary pseudo point). Once the first intersection cycle is complete, additional cycles are generated (as discussed previously) until all the polyline intersection points have been exhausted.

[00107] Fig. 24 illustrates a Category II interaction problem. Once again, we assume that inside is to the right. As indicated in Fig. 15, there exist two intersection and no union boundary-closing cycles. The two intersection boundary-closing cycles are ~~outlined in bold gray~~ lines. The balance of the internal cycles shown ~~outlined in bold gray lines~~ represent additional cycles associated with the intersection product. Sandwiched between these intersection cycles are a series of those portions of the cell which are excluded from the union, ~~outlined by the bold~~

20

~~black~~characterized by regions shown in vertical fill -lines. Identifying the portions of the cell excluded from the union cycle is accomplished by beginning at the first polyline intersection point associated with either the ~~black~~solid or ~~gray~~dashed feature and continuing to accumulate cycles in the manner described previously until all the polyline intersection points have been exhausted.

[00108] Despite subtle differences between the construction of Category I & II and Category III - VI set operation products, the mechanics of the process are obviously quite similar. Once the first cycle has been developed (be it a boundary-closing or internal cycle), the next cycle (for the same Boolean set operation product) begins at the next unused intersection point. The construction process terminates when no unused intersection points remain. Note also that union cycles occur between intersection cycles and conversely, intersection cycles occur between union cycles, regardless of the cell interaction Category. Because of this characteristic of the method of instant invention, rather than independently generating intersection and union products, the method permits efficient construction of both products during a single pass.

[00109] Regions with Included Holes

[00110] Because the tuple ordering convention provides a consistent interpretation of a region's interior, the instant set operation generation procedure handles regions containing included holes without difficulty. Figs. 25-28 provide examples that illustrate the straightforward nature of the analysis. As is evident in these figures, hole boundaries are handled identically to outer region boundaries in terms of set operation generation procedures. Note that in Fig. 27 and 28, the intersection product remains the same, regardless of whether the hole is fully enclosed or only partially enclosed by the quadtree indexing cell.

[00111] Recursive Operation

[00112] Once a given cycle is complete, a determination must be made whether the analysis of that cell is finished. Cell processing is complete when no additional unused polyline intersection points exist. If unused polyline intersection points remain, at least one additional internal cycle must exist. The next remaining cycle begins at the next unused intersection point along the entrance feature indicated in Fig. 15 (actually, either feature may be followed as long as cycles

are developed in ascending tuple order) and continues until the cycle is complete. Note that the number of internal cycles that exist within the cell is highly sensitive to the characteristics of the data sets. Thus, for two or more internal intersection points, the number of internal cycles can not be reliably predicted.

[00113] <u>Final Product Development</u>

[00114] The final set operation product of step 40 (Fig. 6) for two quadtree-represented regions is assembled as the concatenation of the products generated by the three stages of analysis. Class 1 analysis contributes only region interior cells. For set intersection, both Class 2 and Class 3 generate only boundary cell products; for set union, Class 2 generates only interior cells, while Class 3 generates boundary cell products. Under special conditions, the latter can generate interior cells. If the desired set operation product is a vector boundary list (rather than a quadtree-indexed vector representation), the ordered tuple list or lists (in the case of disjoint of embedded set operation products) is generated by a simple quadtree-indexed vector boundary cell-to-vector boundary conversion which effectively strips off the indexing cell representation.

[00115] In all of the examples presented in this document, the area "inside" a region has been assumed to be "to the right" of the ordered tuple boundary list. To maintain this convention, region boundaries use a clockwise tuple ordering. If, on the other hand, tuples are ordered in a counter-clockwise direction, "inside" will be "to the left."

[00116] Fig. 29 is a table that summarizes general characteristics of each canonical form class and reflects the fact that very similar procedures are appropriate for constructing set operation products for both ordering conventions. Columns 1 and 2 are identical to those in Fig. 15. Column 3 identifies whether the total number of polyline intersection points within a cell is odd or even. Column 4 identifies whether the entry/exit pseudo points for a given feature are adjacent (e.g., $\cancel{G}\underline{D}_E$, $\cancel{G}\underline{D}_X$, $B\underline{S}_E$, $B\cancel{\underline{S}}_X$) or whether they alternate (e.g., $\cancel{G}\underline{D}_E$, $B\underline{S}_E$, $\cancel{G}\underline{D}_X$, $B\cancel{\underline{S}}_X$). | Columns 6 and 7 indicate the total number of intersection and union cycles that occur within the cell for the number of polyline intersection points specified in Column 5.

[00117] Referring back to the software flow of Fig. 6, after the final set operation product is calculated in step 40, the software method of the present invention proceeds through step 50

where the set operation product is displayed and/or otherwise output for the user. At step 60 the software preferably releases the memory resources that it used for the foregoing computation. The user is given the option of completing another set operation product at step 70, and if such is desired the program returns to step 30. Otherwise, at step 80 the software releases the memory resources that it used in establishing the bounding box. The user is given the option of adding more features to the problem at step 90, and if such is desired the features are added and the program returns to the beginning (A) for initialization of a new bounding box. Otherwise, the program is done.

[00118] The foregoing method capitalizes on the efficiencies gained by quadtree-indexed vector representation, and gives orders of magnitude improvement in efficiency over current computational methods.

[00119] Efficiency

[00120] In the following explanation, the computational complexity of the proposed set operation generation approach will be compared to traditional computational geometry approaches to prove the efficiencies gained by the latter. When set operations are performed on two regions possessing significantly different sizes, the computational efficiency improvement over traditional computational geometry approaches is on the order of the number of boundary cells associated with the larger of the two regions. Because this number can be quite large, the potential reduction in computational requirements permits the implementation of a wide range of demanding applications that heretofore have been impractical.

[00121] Traditional computational geometry approaches to polygon intersection between two simply-connected regions are known to possess computational complexity of order $m \times n$ (where m is the number of tuples in one region and n is the number of tuples in the other region). Thus, for even moderately large values of m and n, computational requirements tend to be quite high.

[00122] Fig. 30 is a graphical illustration of two regions exhibiting (a) limited boundary cell overlap and (b) two regions exhibiting extensive edge overlap.

[00123] When only a relatively small percentage of the boundary cells of the two regions intersect (as in Fig. 30(a)), the computational requirements of the present approach tends to be rather modest. The worst-case conditions occur when the two regions are of approximately equal physical size and possess significant overlap (see Fig. 30(b)). In this case, most, if not all, the boundary cells of the two regions interact. Thus, unlike traditional set operation generation approaches that tend to be somewhat independent of either the detailed character of the regions or their degree of overlap, the computational complexity of the quadtree-indexed vector based approach is highly sensitive to both considerations. Even the worst case performance can be orders of magnitude better than traditional computational geometry approaches. The number of tuples associated with the quadtree boundary cells for two regions will be defined as m_i and n_j, respectively. Thus, $m = m_i$ and $n = n_j$, where $i = 1,2,...,I$ and $j = 1,2,...,J$. The "worst case" computational complexity of the boundary cell set operation generation method is therefore of order $(m_i \times n_j) = (m_1 \times n_1) + (m_2 \times n_2) + (m_3 \times n_3) + ...$ By rewriting the computational complexity of the conventional computational geometry approach we find:

$$m \times n = (m_1 + m_2 + m_3 + ...) \times (n_1 + n_2 + n_3 + ...) =$$
$$(m_1 \times n_1) + (m_2 \times n_2) \times (m_3 \times n_3)... + (m_1 \times n_2) + (m_1 \times n_3) + (m_2 \times n_1) \times (m_2 \times n_3) + ...$$

[00124] The computational requirements for the computational geometry approach is seen to involve a large number of cross-product terms not present in the proposed approach. As a consequence, the computational requirements associated with the new approach tend to be considerably less than that of traditional formulations. In addition, because the summation $(m_i \times n_j)$ need only be performed over those cells whose boundary cells are shared between the two regions, as stated above, the actual computational complexity will, in general, be considerably less than the worst case analysis predicts.

[00125] To simplify the derivation, we will assume that all boundary cells contain a uniform number of tuples (i.e., $m_i = m/I$ and $n_i = n/J$ for region 1 and region 2, respectively). Based on the worst-case node (cell) count theorem, it can be assumed that the number of interior and boundary quadtree indexing cells of each region are approximately equal. Thus, region 1 will be represented by I interior and I boundary cells; likewise, region 2 will be represented by J interior cells and J boundary cells. In addition, with no loss of generality, we assume that $m = n$. Based

on these assumptions and using the search-based approach for evaluating cell interactions (the region with the lowest cell count controls the number of required operations), the worst case computational complexity for the three classes of cell intersection are as follows:

$$\text{(interior, interior) cell} = I$$

$$\text{(intrerior, boundary) cell} = I$$

$$\text{(boundary, boundary) cell} = I \times (m/I \times n/J) = (m \times n)/J$$

[00126] The latter expression arises because, for each boundary cell in the physically smaller of the two regions, all possible line segment intersections between, the piecewise continuous line segments associated with two features must be computed.

[00127] If $1 \ll m$ n, it becomes readily apparent that (m x n) / J >> I. Thus, quite clearly, the (boundary cell) x (boundary cell) component of the computation (sub problem 3) dominates the complexity of the present approach. Therefore, the computational complexity of the present approach is on the order of $(m \times n)/J$ and the "worst case" relative efficiency (with respect to a computational geometry-based approach), E_R, is:

$$E_R = (m \times n)/(m \times n)/J = J = n/n_j$$

[00128] In other words, the relative efficiency of the present approach with respect to the conventional computational geometry approach is equal to the number of indexing cells in the larger of the two regions (i.e., J). Equivalently, the relative efficiency is proportional to the number of tuples in the larger of the two regions (i.e., n). As the "size" of the larger region increases, so does the relative efficiency. If the number of points in the larger region is increased by two orders of magnitude, for instance, the relative efficiency will increase by approximately that same proportion.

[00129] When little overlap exists between the two regions (such as in Fig. 30(a)), the number of boundary cells (and their associated tuples) is much less than I. Consequently, overall performance will be better than the above performance bound predicts.

[00130] The "worst-case" performance of the present approach occurs when all boundary cells of the two regions intersect. In this situation, $I = J$ and can be replaced by n / I. Under these conditions, the "worst-case." performance improvement reduces to

$$E_R = n / n / I = I$$

[00131] As a simple example, suppose $I = 100$. Thus, the "worst case" relative efficiency improvement over traditional computational geometry algorithms is on the order of 100 (i.e., $I = 100$). If the indexing cell size was further refined so that both regions contained 1000 boundary cells (i.e., $I = 1000$), the present approach could be as much as one thousand times more efficient than traditional computational geometry-based intersection algorithms.

[00132] Performance tests conducted using the software implementation of the generation methodology corroborate these theoretically derived performance bounds.

[00133] Having now fully set forth the preferred embodiments and certain modifications of the concept underlying the present invention, various other embodiments as well as certain variations and modifications of the embodiments herein shown and described will obviously occur to those skilled in the art upon becoming familiar with said underlying concept. It is to be understood, therefore, that the invention may be practiced otherwise than as specifically set forth herein.

ABSTRACT

Performing Boolean operations among two regions; regions and results represented as vector tuples. Establish indexing cells about regions and classify by type of interaction between regions, e.g., (boundary, boundary). For each (boundary, boundary) cell, define pseudo-points at each boundary entrance/exit. Categorize each (boundary, boundary) cell on relationship of its pseudo-points. Identify starting points along boundaries based on: categorization, operation, and interior convention. Accumulate results cycling from a starting point, along region boundary. Upon encountering each intersection, proceed along the other region boundary. Upon encountering a cell edge, proceed along the cell edge in the direction consistent with the interior convention. Where a starting point or intersection between region boundaries remains untraversed, accumulate results in tracing a similar cycle along untraced boundaries. Discard duplicate/cell-edge only tuples. After traversing all starting points/intersections of boundaries, determine results for other cell types, and concatenate with those for (boundary, boundary) cells.